WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Friday, December 02, 2005

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC; PLUR=YES; OP=ADJ	
☐	L13	L7 and (prepar$ same (instance or operation))	24
☐	L12	L7 and prepar$	51
☐	L11	717/158-159.ccls. and (profil$ or instance).ab.	79
☐	L10	717/154.ccls. and (profil$ or instance).ab.	15
☐	L9	717/130-131.ccls. and (profil$ or instance).ab.	89
☐	L8	L7 and (hot?spot or hotspot)	17
☐	L7	L6 and L1	158
☐	L6	L4	784
		DB=EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L5	L4	6
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L4	L3 and optimiz$	790
☐	L3	L2 and virtual machine	1511
☐	L2	Java and (profile or profiling)	9176
☐	L1	(717/124 |717/125 |717/126 |717/127 |717/128 |717/129 |717/130 |717/131 |717/132 |717/133 |717/134 |717/135 |717/136 |717/137 |717/138 |717/139 |717/140 |717/141 |717/142 |717/143 |717/144 |717/145 |717/146 |717/147 |717/148 |717/149 |717/150 |717/151 |717/152 |717/153 |717/154 |717/155 |717/156 |717/157 |717/158 |717/159 |717/160 |717/161 |717/162 |717/163 |717/164 |717/165 |717/166 |717/167).ccls.	5615

END OF SEARCH HISTORY

Hit List

First Hit | Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 7 of 7 returned.

☐ 1. Document ID: US 20040111714 A1

L6: Entry 1 of 7 File: PGPB Jun 10, 2004

PGPUB-DOCUMENT-NUMBER: 20040111714
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20040111714 A1

TITLE: Dynamic division optimization for a just-in-time compiler

PUBLICATION-DATE: June 10, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Shi, Xiaohua	Beijing	CA	CN
Lueh, Guei-Yuan	San Jose		US
Ying, Zhiwei	Beijing		CN

US-CL-CURRENT: 717/148; 717/127, 717/153

CLAIMS:

What is claimed is:

1. A method for improving the performance of a dynamic compiler, comprising: receiving a first code; determining a strategy for optimizing a segment of the first code; optimizing the segment of the first code using the determined optimization strategy; and outputting a second code, representing the optimized first code.

2. The method of claim 1, wherein the first code represents a computer programming code in a high-level programming language.

3. The method of claim 2, wherein the high-level programming language comprises a compiled Java code.

4. The method of claim 1, wherein determining the strategy comprises dynamically determining a best strategy available to the compiler to optimize the segment of the first code according to characteristics of the segment.

5. The method of claim 1, wherein optimizing the segment of the first code comprises converting the segment in a high-level language to a set of lower-level computing instructions native to an underlying computing architecture, selecting a best optimization implementation, and optimizing the set of lower-level computing instructions based on the determined optimization strategy.

38. The system of claim 33, wherein the divisor profiling mechanism comprises: a creation component to create a new entry in a divisor cache, if a divisor appears for the first time; an initialization component to initialize at least a flag field and an optimization parameter field of the newly created entry for the divisor in the divisor cache; and a counting component to increment the number of occurrences of the divisor and record the new number in the divisor counter field of the entry for the divisor in the divisor cache.

39. The system of claim 38, wherein the divisor profiling mechanism further comprises: a component to determine whether a divisor becomes invariant for the first time during runtime by comparing the number of occurrences of the divisor with a pre-set trigger number; a component to select a best optimization implementation for a division code with the divisor; and a component to send requests to the optimization preparation mechanism to prepare optimization parameters for the divisor, if the divisor becomes invariant for the first time during runtime.

40. The system of claim 39, wherein the divisor profiling mechanism further comprises a component to store the prepared optimization parameters in the divisor cache for the divisor, and to replace the value of the divisor's flag field with the divisor in the divisor cache.

41. The system of claim 33, wherein the optimization preparation mechanism comprises: a component to prepare optimization parameters required by the selected optimization implementation; a component to pass optimization parameters to the divisor profiling mechanism to update a divisor cache, and to the selected optimization implementation to invoke the selected optimization implementation.

☐ 2. Document ID: US 20030105942 A1

L6: Entry 2 of 7 File: PGPB Jun 5, 2003

PGPUB-DOCUMENT-NUMBER: 20030105942
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030105942 A1

TITLE: Aggressive prefetch of address chains

PUBLICATION-DATE: June 5, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Damron, Peter C.	Fremont	CA	US
Kosche, Nicolai	San Francisco	CA	US

US-CL-CURRENT: 712/216; 712/207

CLAIMS:

What is claimed is:

1. In a scheduler for computer code wherein certain operations are likely to stall

execution of the computer code and thereby provide latency for completion of one or more pre-executable operations, a method of scheduling certain of the operations, the method comprising: for one or more sequences of operations that follow a speculation boundary and that define respective dependency chains, including pre-executable operations, which lead to likely stalls, representing speculative copies thereof as duplicate chains; and scheduling operations of the computer code, wherein the scheduling of operations from the duplicate chains is performed without regard to dependence of respective original operations on the speculation boundary, thereby scheduling certain of the operations above the speculation boundary into position preceding at least one of the operations likely to stall execution of the computer code.

2. A method, as recited in claim 1, wherein the likely stalls include likely cache misses.

3. A method, as recited in claim 1, wherein the dependency chains include address chains leading to memory access operations likely to miss in a cache.

4. A method, as recited in claim 1, wherein the pre-executable operations include prefetch instructions.

5. A method, as recited in claim 1, wherein the pre-executable operations include speculative operations.

6. A method, as recited in claim 1, wherein the operations likely to stall execution include memory access instructions.

7. A method, as recited in claim 1, wherein the operations likely to stall execution include operations selected from the set of: a load operation; first use of a load operation; a store operation; a branch operation; a multi-cycle computational operation; an iterative or recursive operation; a communications operation; an input/output (I/O) operation; a synchronization operation; and a co-processor operation.

8. A method, as recited in claim 1, wherein the speculation boundary is defined by one of: a store operation; a branch operation; a join operation; an iterative or recursive operation; a communications operation; an input/output (I/O) operation; a synchronization operation; and a co-processor operation.

9. A method, as recited in claim 1, further comprising: inserting the pre-executable operations into the computer code.

10. A method, as recited in claim 1, further comprising: profiling the computer code to identify the likely stalls.

11. A method, as recited in claim 1, further comprising: upon reaching the speculation boundary, deleting unscheduled operations of the duplicate chains and continuing to schedule respective original operations.

12. A method, as recited in claim 1, further comprising: deleting from the original operations, pre-executable operations for which a respective speculative copy is scheduled.

13. A method of hiding latency in computer code wherein certain operations thereof are likely to stall execution, the method comprising: identifying sequences of operations that define respective original dependency chains that lead to likely stalls and for at least some of the identified sequences, representing duplicate

prefetch instruction in the execution sequence.

44. The computer program product of claim 42, further comprising: a martyr instruction that follows the speculative load instruction and the prefetch instruction which, upon execution, provides at least a portion of a latency therefor.

45. The computer program product of claim 42, prepared by a program scheduler that inserts prefetch instructions into the execution sequence and schedules speculative duplicates of at least some load instructions together with corresponding prefetch instructions above speculative boundaries therein.

46. The computer program product of claim 42, wherein the one or more computer readable media are selected from the set of a disk, tape or other magnetic, optical, semiconductor or electronic storage medium and a network, wireline, wireless or other communications medium.

47. An apparatus comprising: a code preparation facility for transforming schedulable code into scheduled code; and means for scheduling speculative copies of operations that form dependency chains that lead to a likely stall, the scheduling placing the speculative operations above a preceding at least one other operation that is itself likely to stall, thereby hiding in the scheduled code latency of the speculative operations.

48. The apparatus of claim 47, further comprising: means for inserting pre-executable operations into the schedulable code, wherein at least some of the pre-executable operations are scheduled be the scheduling means as the speculative operations for which latency is hidden.

49. The apparatus of claim 47, further comprising: means for identifying likely-to-stall operations of schedulable code.

☐ 3. Document ID: US 20030101443 A1

L6: Entry 3 of 7 File: PGPB May 29, 2003

PGPUB-DOCUMENT-NUMBER: 20030101443
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030101443 A1

TITLE: Technique for associating execution characteristics with instructions or operations of program code

PUBLICATION-DATE: May 29, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Kósche, Nicolai	San Francisco	CA	US
Aoki, Christopher P.	Los Altos	CA	US
Damron, Peter C.	Fremont	CA	US

US-CL-CURRENT: 717/158

CLAIMS:

What is claimed is:

1. A code preparation method comprising: identifying at least one operation in first executable instance of code; executing the first executable instance and responsive to detection of an execution event, associating a corresponding execution characteristic with a corresponding identified one of the operations; and preparing a second executable instance of the code based, at least in part, on the association between the execution characteristic and the identified operation.

2. The method of claim 1, wherein the operation identification is consistent between the first executable instance and the preparation of the second executable instance.

3. The method of claim 2, wherein the consistency of operation identification is maintained from preparation of the first executable instance to preparation of the second executable instance.

4. The method of claim 1, wherein same unique identification numbers are assigned to corresponding operations of the first executable and the second executable.

5. The method of claim 4, wherein the execution characteristic is associated with the unique identification number.

6. The method of claim 4, wherein the unique identification numbers and their assignment to operations are maintained throughout any optimizations or code transformations performed in preparation of the first executable.

7. The method of claim 6, wherein the maintenance of the unique identification number assignments include further assigning the unique identification number to a copy when an operation is copied as part of a code transformation or optimization.

8. The method of claim 6, wherein the maintenance of the unique identification number assignments includes removing an assignment when the assigned operation is removed as part of a code transformation or optimization.

9. The method of claim 1, wherein the associating of the corresponding execution characteristic includes encoding aggregated hardware event information in an extended definition of an instruction instance for use in the preparation of the second executable instance.

10. The method of claim 1, wherein the identified operation is a memory access instruction.

11. The method of claim 1, wherein the execution characteristic includes a cache miss likelihood.

12. The method of claim 1, wherein the preparation includes inserting one or more prefetch operations in the code prior to the identified operation to exploit latency provided by servicing of a cache miss by the identified operation.

13. The method of claim 1, further comprising: preparing the first executable instance.

14. The method of claim 13, wherein the preparation of the first executable

PGPUB-DOCUMENT-NUMBER: 20020059568
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020059568 A1

TITLE: Program compilation and optimization

PUBLICATION-DATE: May 16, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Kawahito, Motohiro	Sagamihara-shi		JP
Ogasawara, Takeshi	Tokyo-to		JP
Komatsu, Hideaki	Yokohama-shi		JP

US-CL-CURRENT: 717/151

CLAIMS:

What is claimed is:

1. A compiler for converting source code for a program written in a programming language into object code in a machine language, comprising: an optimization execution unit for performing an optimization process for an object program written in a machine language; and a program modification unit for modifying said object program in order to absorb a difference in content between the point of origin of an exception process, which occurs in response to the execution of a command in said object program, and a location whereat said exception process is performed.

2. The compiler according to claim 1, wherein, if there is a difference in content between the point of origin of an exception process, which occurs in response to the execution of a command in said object program, and a location whereat said exception process is performed, said program modification unit generates compensation code to compensate for said difference, and inserts said compensation code into said object program.

3. The compiler according to claim 1, wherein said program modification unit includes: a pre-processor for, before said optimization execution unit performs said optimization process, employing a Try node to examine a command that may cause an exception process in said object program to determine whether an exception process has occurred, and a Catch node for performing an inherent process when it is found an exception process has occurred; and a post-processor for examining, in said object program that has been optimized by said optimization execution unit, said command that may cause an exception process to determine whether a difference in content exists between said command that may cause said exception process and a location whereat said exception process is performed, and for, when a difference exists, generating in said Catch node a compensation code, to be used to compensate for said difference, and a code for, after said compensation code is obtained, moving program control to said location whereat said exception process is performed.

4. The compiler according to claim 1, wherein, before said optimization execution unit performs said optimization process in said object program, said program modification unit divides said command that may cause an exception process into a

has occurred, and that includes a command for, when an exception process has occurred, moving program control to a portion whereat said exception process is performed; and a process for, when a difference in content exists between the point of origin of said exception process and said portion whereat said exception process is performed, generating in said basic block compensation code for compensating for said difference.

16. A program transmission apparatus comprising: storage means for storing a program that permits a computer to perform a process for preparing a basic block that includes a portion for examining a command, in an object program, that may cause an exception process, in order to determine whether an exception process has occurred, and that includes a command for, when an exception process has occurred, moving program control to a portion whereat said exception process is performed, and a process for, when a difference in content exists between the point of origin of said exception process and said portion whereat said exception process is performed, generating in said basic block compensation code for compensating for said difference; and transmission means for reading said program from said storage means and for transmitting said program.

☐ 5. Document ID: US 5404555 A

L6: Entry 5 of 7 File: USPT Apr 4, 1995

US-PAT-NO: 5404555
DOCUMENT-IDENTIFIER: US 5404555 A

TITLE: Macro instruction set computer architecture

DATE-ISSUED: April 4, 1995

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Liu; Dali	Beijing			CN

US-CL-CURRENT: 712/36; 712/202

CLAIMS:

What is claimed is:

1. A macro-instruction set computer achictecture comprising:

main memory means for storing system softwares of the computer, instructions and user programs;

first memory means for storing preparatory data for operation, intermediate results of operation and final results of completed operation, and operating in the form of a stack;

second memory means for storing a break point address of subprograms and address for recovery of a break point while returning from call, and operating in the form of a stack; and

Web Results **1** - **10** of about **346** for **Optimized compiler profile JIT preparation hotspot**. (**0.24** seconds)

Web Pages Related to Compiling Java into Native Code
As a result, **JIT** and **hot spot** techniques resulted in Java performance that ...
Profile driven **optimization** is currently used by some IA-64 **compilers**. ...
www.bearcave.com/software/java/comp_java.html - 42k - Cached - Similar pages

JOT: Journal of Object Technology - E-Bunny: A Dynamic **Compiler** ...
to **optimize** programs before execution. Just-in-time (**JIT**) **compilation** consists of the ... The Java **Hotspot** VM dynamic **compiler** applies several classical ...
www.jot.fm/issues/issue_2005_01/article2 - 84k - Cached - Similar pages

[PDF] E-Bunny: A Dynamic **Compiler** for Embed- ded Java Virtual Machines
File Format: PDF/Adobe Acrobat - View as HTML
to **optimize** programs before execution. Just-in-time (**JIT**) **compilation** ...
Optimization System (AOS). As Java **Hotspot** VM, the features of IBM Jalape˜no ...
www.jot.fm/issues/issue_2005_01/article2.pdf - Similar pages

Java performance tuning tips
Before manual tuning, **HotSpot** VMs are often faster than **JIT** VMs. ... Use the Java **compiler**?s **optimization** flag (javac -O); **Profile** the application (using ...
www.javaperformancetuning.com/tips/rawtips.shtml - 419k - Cached - Similar pages

[PDF] Accelerating Embedded Java for Mobile Devices
File Format: PDF/Adobe Acrobat
posed approach to dynamic **compilation**. **JIT** con- ... slow **optimizing compiler** that produces a high ... [7] Sun MicroSystems, The Java **HotSpot** Performance ...
ieeexplore.ieee.org/iel5/ 35/32334/01509971.pdf?arnumber=1509971 - Similar pages

[PDF] A Survey of Adaptive **Optimization** in Virtual Machines
File Format: PDF/Adobe Acrobat
a fast **JIT compiler**, and a slow "traditional" **compiler** adapted. for use as a **JIT**. ... a threshold, the system initiated a new **profile/optimization** ...
ieeexplore.ieee.org/iel5/ 5/30187/01386662.pdf?arnumber=1386662 - Similar pages

[PDF] A Dynamic **Compiler** for Embedded Java Virtual Machines
File Format: PDF/Adobe Acrobat
optimize programs before execution. Just-in-time (**JIT**) **compilation** consists of ... No more details are provided about the CLDC **Hotspot**. dynamic **compiler**. ...
portal.acm.org/ft_gateway.cfm?id=1071584&type=pdf - Similar pages

[PDF] **Optimizations** for a Java Interpreter Using Instruction Set Enhancement
File Format: PDF/Adobe Acrobat - View as HTML
any of the **optimizations** described in this paper. We also. show running times for Sun's **HotSpot** 1.4.2 mixed-mode in-. terpreter and **JIT compiler**, ...
https://www.cs.tcd.ie/publications/ tech-reports/reports.05/TCD-CS-2005-61.pdf - Similar pages

[PDF] IBM Research Report
File Format: PDF/Adobe Acrobat - View as HTML

The **HotSpot** Server **compiler** [38] reports similar techniques. ... evaluate the **optimizations** in the **JIT** for the IBM DK for Java, and report that the cheapest ...
www.research.ibm.com/people/d/dgrove/papers/RC23143.pdf - Similar pages

[PDF] Marmot: An **Optimizing Compiler** for Java
File Format: PDF/Adobe Acrobat - View as HTML
While existing interpreter and just-in-time-**compilation** (**JIT**) based Java ...
In **preparation**, 1999. [Gup90]. Rajiv Gupta. A fresh lookat **optimizing** array ...
research.microsoft.com/~dtarditi/dist/marmot.pdf - Similar pages

Try searching for **Optimized compiler profile JIT preparation hotspot** on Google Book Search

Gooooooooogle ▶
Result Page: 1 2 3 4 5 6 7 8 9 10 **Next**

Optimized compiler profile JIT prepa Search

Search within results | Language Tools | Search Tips | Dissatisfied? Help us improve

Google Home - Advertising Programs - Business Solutions - About Google

Search Results Results **1 - 10** of about **15** for **Optimization compiler profile JIT preparation hotspo**

eb Pages Related to Compiling Java into Native Code
eb Pages Related to Compiling the Java Programming Language. Forward. This web page was originally writter
ıen, in some cases dramatically. ... In Time **compiler** (**JIT**) which compiles ... **optimizations** in **preparation** for g
ıiled mini-supercomputer company). **Profile** driven **optimization** ...
ww.bearcave.com/software/java/comp_java.html - 42k - Cached - More from this site - Save - Block

WN: Sun yanks FreeBSD's Java license
better it includes a **compiler** for the Java language ... a **JIT**, nor does it **profile** code as it runs for further **optimi**
/M ...
n.net/Articles/118461 - 61k - Cached - More from this site - Save - Block

ava performance tuning tips
lock acquisition required in **HotSpot** JVMs (when they have about ... in different test runs; Analyze the algorithm
ıther than by hand ...
ww.javaperformancetuning.com/tips/rawtips.shtml - 429k - Cached - More from this site - Save - Block

ıil Hansen's Java URLs
Exam **Preparation**: Top-level ... **Optimization** @CMU. Just-in-Time **Compilers**. **JIT compiler** ... **HotSpot** VM.
onfiguration (CLDC) Foundation **Profile** ...
ww.javamug.org/mainpages/Java.html - 400k - Cached - More from this site - Save - Block

developers.net
TUTORIALS WHITEPAPERS NEWS. MY **PROFILE** SITEMAP SEARCH TOP SEARCHES HOT ... new Object(
ɔportunities for **optimization** (at the cost of some ...
ww.developers.net/blogs/view/subcat/Java - 525k - Cached - More from this site - Save - Block

OT: Journal of Object Technology - E-Bunny: A Dynamic **Compiler** for Embedded Java Virtua
Just-in-time (**JIT**) compilation consists of the dynamic compilation ... The Java **Hotspot** VM dynamic **compiler** a
ptimization is that it produces ...
ww.jot.fm/issues/issue_2005_01/article2 - 85k - Cached - More from this site - Save - Block

ttp://utnubu.alioth.debian.org/missing-packages-non-orphaned
a dot for the **hotspot** of each cursor - The main ... Haskell Compilation system (GHC). GHC is a **compiler** for H
36 ...
:nubu.alioth.debian.org/missing-packages-non-orphaned - 525k - Cached - More from this site - Save - Block

ink List Labor - US Vereinigte Staaten von Amerika, États-Unis, United States of America - Inf
nk List Labor - US Vereinigte Staaten von Amerika, États-Unis, United States of America - Informatik, IV, PC, W
ww.etymologie.info/~l/u_/us-inform.html - 255k - Cached - More from this site - Save - Block

e medical,best ge medical - ge medical
E MEDICAL. Site about ge medical. A lot of information about ge medical. Links to ge medical sites and articles.
ɘ-medical.metka.rzeszow.pl - 290k - Cached - More from this site - Save - Block

ttp://www.rivier.edu/faculty/amoreira/web/cs574a/gl4java/Gl4Java/CHANGES.txt
context switching **optimization** to an automatical **optimization** of context ... in JVM >= 1.3 (+ **hotspot**) !! This ta

AVA_**COMPILER**=). The **JIT** forces ...
ww.rivier.edu/faculty/amoreira/web/cs574a/gl4java/Gl4Java/CHANGES.txt - More from this site - Save - Block

r to show you the most relevant results, we have omitted some entries very similar to the ones already displayed
ike, you can repeat the search with the omitted results included.
